# HW2: Model Explanability

## 1. Load data for modeling. This data represents taxi rides in NY (from a Kaggle competition)

In [1]:
```
import math
import scipy
import warnings
import numpy as np
import pandas as pd
import matplotlib
import seaborn as sns
import xgboost as xgb
import matplotlib.pylab as plt
import pyarrow

from sklearn.ensemble import RandomForestRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, median_absolute_error

matplotlib.use('nbagg');
warnings.filterwarnings("ignore");
%matplotlib inline
```

In [2]:
```
# Download the data (public github)
!wget -N https://github.com/rfox12-edu/explainability-sandbox/raw/main/x_train.parquet.gzip
!wget -N https://github.com/rfox12-edu/explainability-sandbox/raw/main/x_test.parquet.gzip
!wget -N https://github.com/rfox12-edu/explainability-sandbox/raw/main/y_train.parquet.gzip
!wget -N https://github.com/rfox12-edu/explainability-sandbox/raw/main/y_test.parquet.gzip
```

```
zsh:1: command not found: wget
zsh:1: command not found: wget
zsh:1: command not found: wget
zsh:1: command not found: wget
```

In [3]:
```
# Load the data into pandas dataframes
x_train = pd.read_parquet('x_train.parquet.gzip')
x_test = pd.read_parquet('x_test.parquet.gzip')
y_train = pd.read_parquet('y_train.parquet.gzip')
y_test = pd.read_parquet('y_test.parquet.gzip')
```

## 2. Model Build

### Random Forest Regression

In [4]:
```
regr_rf = RandomForestRegressor(max_features='sqrt', min_samples_leaf = 4,
    min_samples_split = 3, n_estimators = 40, n_jobs = -1)
regr_rf.fit(x_train, y_train)
```

Out[4]:
```
▾ RandomForestRegressor
RandomForestRegressor(max_features='sqrt', min_samples_leaf=4,
                      min_samples_split=3, n_estimators=40, n_jobs=-1)
```

In [5]:
```
y_train_pred_rf = regr_rf.predict(x_train)
mse_train_rf = mean_squared_error(y_train, y_train_pred_rf)

y_pred_rf = regr_rf.predict(x_test)
mse_rf = mean_squared_error(y_test, y_pred_rf)

print('Train MSE:' , mse_train_rf)
print('Test MSE:  ', mse_rf)
```

```
Train MSE: 83.39716525837753
Test MSE:  165.80871060868736
```

### XGboost Regression

In [6]:
```
regr_xgb = xgb.XGBRegressor(
    learning_rate=0.1, n_estimators=1000, max_depth=3, min_child_weight=3,
    gamma=0, subsample=0.8, reg_alpha=200, reg_lambda=200, colsample_bytree=0.8, n_jobs=-1
)
regr_xgb.fit(x_train, y_train)
```

Out[6]:

```
XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.8, device=None, early_stopping_rounds=None
,
             enable_categorical=False, eval_metric=None, feature_types=Non
e,
             gamma=0, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None
,
```

In [7]:

```python
# Predicting on train & test data using our trained XgBoost regressor model
y_train_pred_xgb = regr_xgb.predict(x_train)
mse_train_xgb = mean_squared_error(y_train, y_train_pred_xgb)

y_pred_xgb = regr_xgb.predict(x_test)
mse_xgb = mean_squared_error(y_test, y_pred_xgb)

print('Train MSE:' ,mse_train_xgb)
print('Test MSE: ', mse_xgb)
```

```
Train MSE: 158.24951524443995
Test MSE:  168.18855740911232
```

### Feed forward NN: MLP

In [8]:

```python
regr_mlp = MLPRegressor(
        hidden_layer_sizes=[50, 25],
        activation='relu',
        solver='adam',
        early_stopping=True,
        random_state=33
)
regr_mlp.fit(x_train, y_train)
```

Out[8]:

```
MLPRegressor
MLPRegressor(early_stopping=True, hidden_layer_sizes=[50, 25], random_state=33)
```

In [9]:

```python
y_train_pred_mlp = regr_mlp.predict(x_train)
mse_train_mlp = mean_squared_error(y_train, y_train_pred_mlp)

y_pred_mlp = regr_mlp.predict(x_test)
mse_mlp = mean_squared_error(y_test, y_pred_mlp)

print('Train MSE:' , mse_train_mlp)
print('Test MSE: ', mse_mlp)
```

```
Train MSE: 166.36158474862745
Test MSE:  163.90548439517627
```

### Q1 (5pts): Which model is the most over-fit to its training data?

- Random Forest among the models has the biggest difference so it is overfitting.

### Q2 (5pts): Is AUC an appropriate metric to evaluate these models? Why or why not?

- No AUC is primarily used in multiclass/binary classification models compared to these regression models. The goal is to divide multiple classes compared to predicting outcomes that can be continuous.

# Global Explanability

## Random Forest Feature Importance

### Q3 (5pts): Why is it important to look the overall predictive power of a model before looking at feature importance for that model?

- Poor predictive power reflects how a model perfoms on unseen data and can indicate a model is lacking or flawed in some way, making the feature importance less accurate in their interpetation.

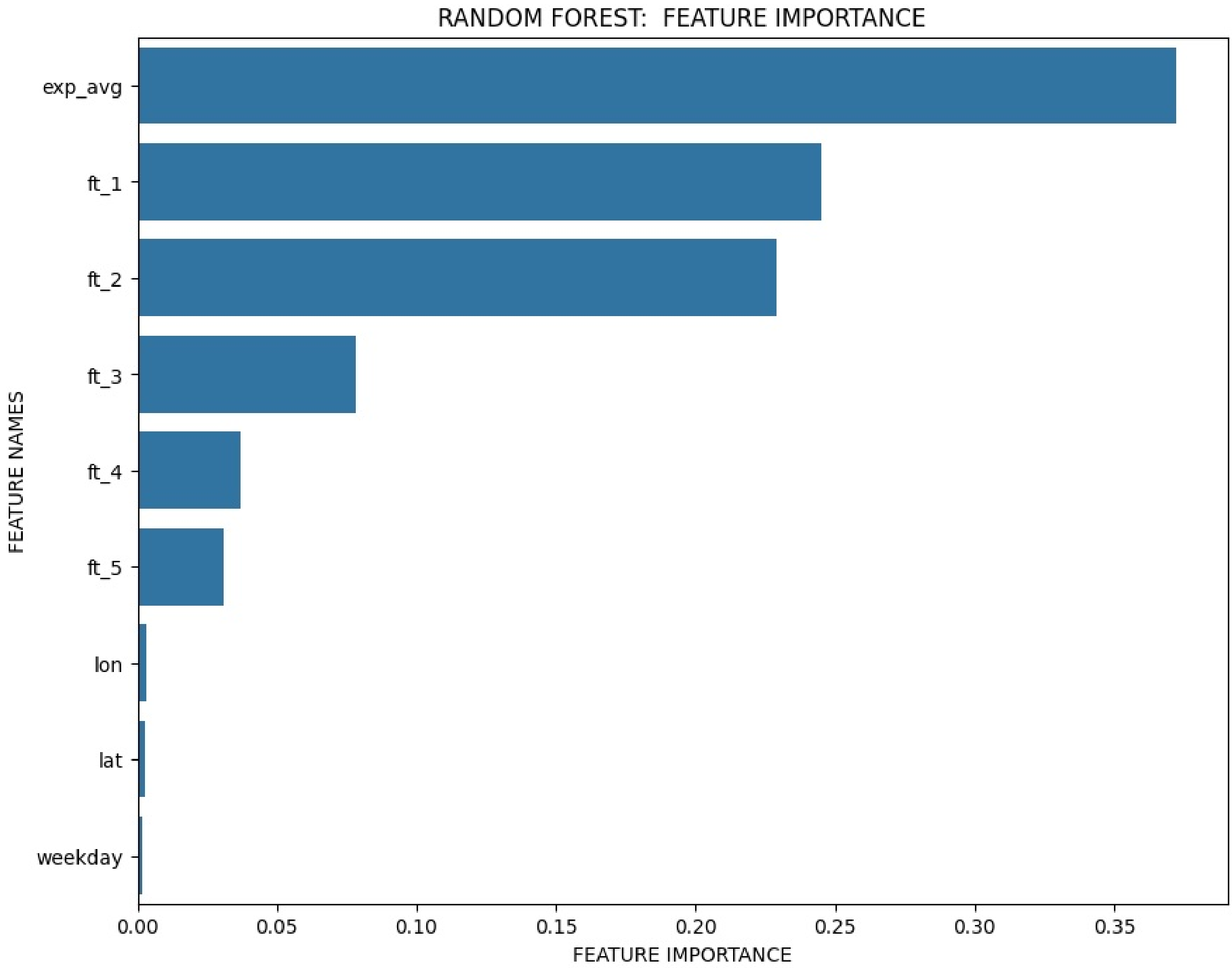
Q4 (5pts): How could you validate that these regression models are providing any predictive power at all? [you do not have to write code to answer this question]

- Validation can be very helpful by checking that it is making meaningful prediction and not overfitting/underfitting our data. Using Train-Test Split is one method to assess how generalized our model is while another is K-fold validation.

In [10]:
```
def plot_feature_importance(importance,names,model_type):

    #Create arrays from feature importance and feature names
    feature_importance = np.array(importance)
    feature_names = np.array(names)

    #Create a DataFrame using a Dictionary
    data={'feature_names':feature_names,'feature_importance':feature_importance}
    fi_df = pd.DataFrame(data)

    #Sort the DataFrame in order decreasing feature importance
    fi_df.sort_values(by=['feature_importance'], ascending=False,inplace=True)

    #Define size of bar plot
    plt.figure(figsize=(10,8))
    #Plot Searborn bar chart
    sns.barplot(x=fi_df['feature_importance'], y=fi_df['feature_names'])
    #Add chart labels
    plt.title(model_type + ':  FEATURE IMPORTANCE')
    plt.xlabel('FEATURE IMPORTANCE')
    plt.ylabel('FEATURE NAMES')
```

In [11]:
```
def permutation_based_feature_importance(x_test, y_test, initial_mse, model):

    # Initialize an array to store feature importances
    feature_importances = np.zeros(x_test.shape[1])

    # Number of permutation iterations (you can adjust this value)
    num_iterations = 100

    # Calculate feature importance by permuting one feature at a time
    ####################### CODE HERE ############################
    for feature in range(x_test.shape[1]):
        print('Permuting feature ',feature + 1)
        # Copy the original test data
        x_test_permuted = x_test.copy()

        # Shuffle the values of the current feature
        permuted_column = x_test_permuted.iloc[:, feature]
        np.random.shuffle(permuted_column)
        x_test_permuted.iloc[:, feature] = permuted_column

        # Calculate the accuracy with the permuted feature
        permuted_mse = mean_squared_error(y_test, model.predict(x_test_permuted))

        # Calculate the drop in accuracy and store it as feature importance
        feature_importances[feature] = initial_mse - permuted_mse

    # Normalize the feature importances
    feature_importances /= feature_importances.sum()

    # Get the names of the features (assuming X is a DataFrame)
    feature_names = x_test.columns

    # Sort features by importance
    sorted_idx = np.argsort(feature_importances)

    return feature_importances[sorted_idx]
```

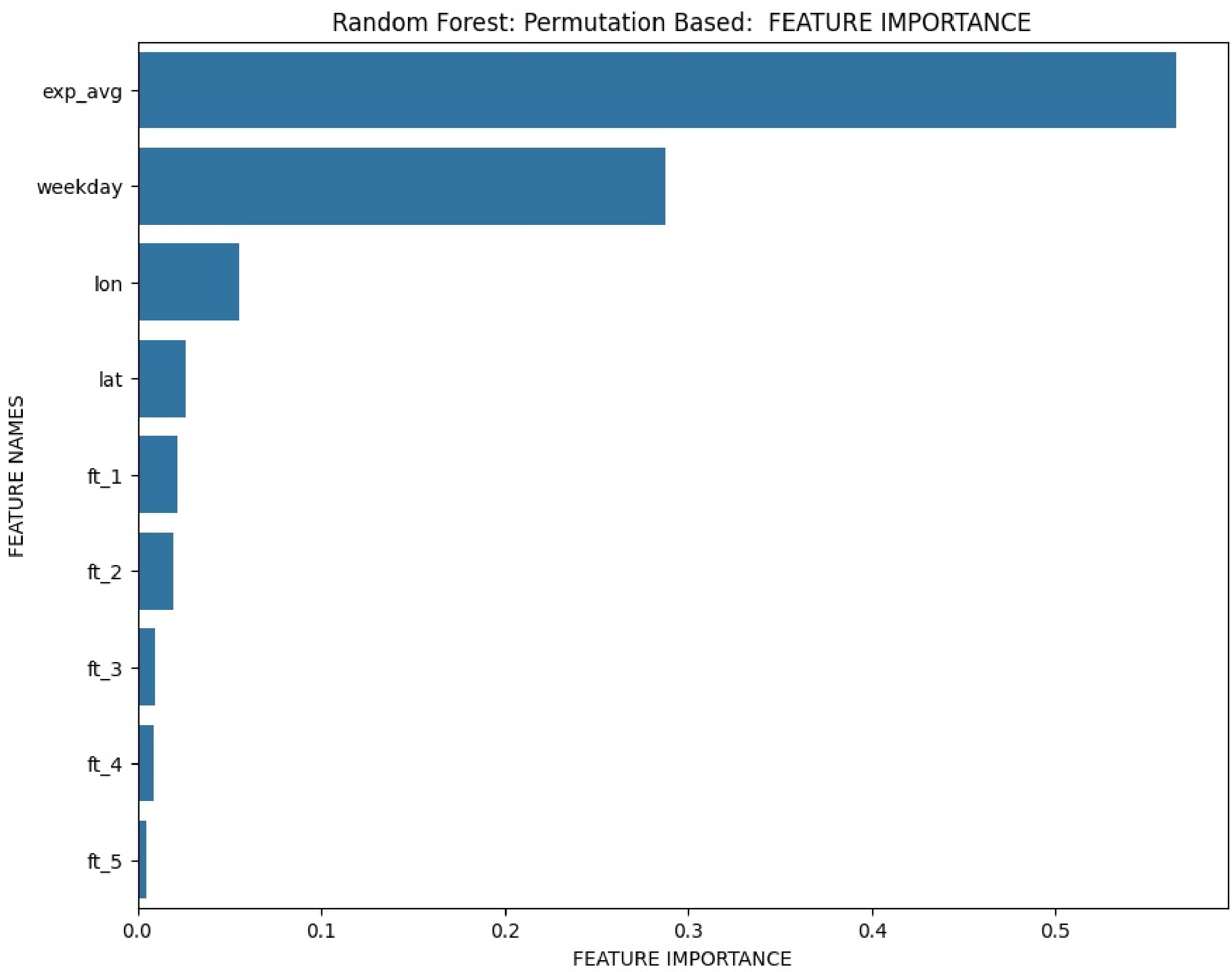
In [12]:
```
plot_feature_importance(regr_rf.feature_importances_,x_train.columns,'RANDOM FOREST')
```




In [13]:
```
permutation_based_feature_importance_rf = permutation_based_feature_importance(x_train, y_train, mse_train_rf,
```

```
Permuting feature  1
Permuting feature  2
Permuting feature  3
Permuting feature  4
Permuting feature  5
Permuting feature  6
Permuting feature  7
Permuting feature  8
Permuting feature  9
```

In [14]:
```
plot_feature_importance(permutation_based_feature_importance_rf,x_train.columns,'Random Forest: Permutation Base
```




Q5 (5pts): Is the `weekday` feature important to the `regr_rf` model?

- In a normal gini importance model, it has little importance but when permutated (measures the decrease in model performance when a feature's values are randomly shuffle), it plays a large part in the feature importance. Weekday in this regard, is important to the underlying features different from nomral methods.

Q6 (5pts): Is the `weekday` feature important to predicting the target?

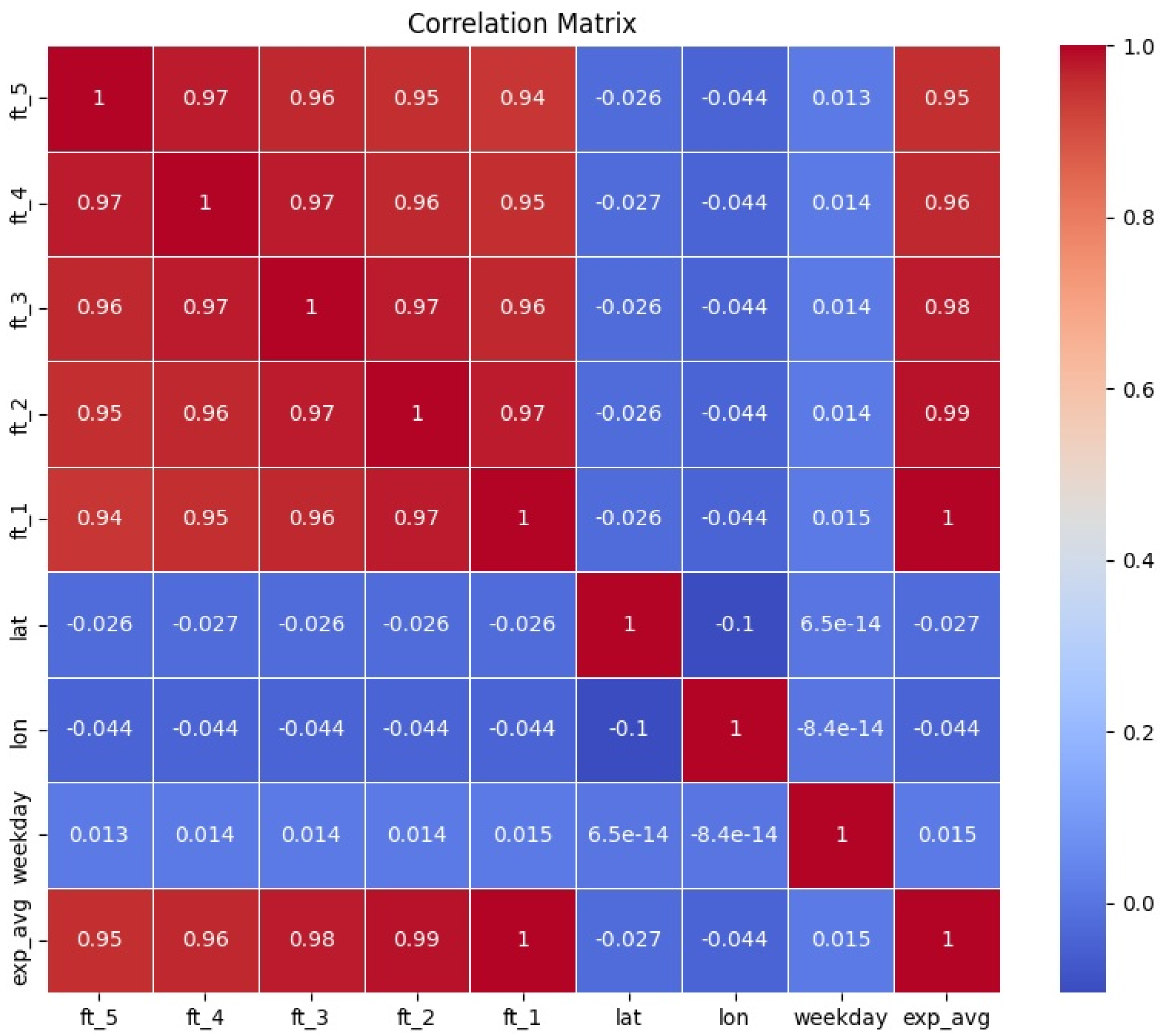
- I think it is important, this is supported by the permutated feature importance as it tends to give a more reliable measure of each features predictive power. It being the second highest indicates that the model needs weekday for its predictive capabilities.

Q7 (10pts): Is the `.feature_importances_` of the `regr_rf` model reliable? Why or why not? hint

- It is not always more reliable as the feature_importances_ method is based on the gini impurity or mean decrease impurity. Can inflate the importance of high-cardinality features and those that appear more frequently in the training data and suffer overfitting if computed on training data.

Q8 (5pts): Are there any highly correlated features in this dataset? Which ones?

In [17]:
```
# Calculate the correlation matrix
corr_matrix = x_train.corr()

# Plotting the heatmap
plt.figure(figsize=(10,8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```




- There are many highly correlated features in the dataset, the following are higly correlated:
- exp_avg, ft_1, ft_2, ft_3, ft_4, and ft_5.

Q9 (5pts): What is the impact of highly correlated features on the analysis of feature importance for these models?

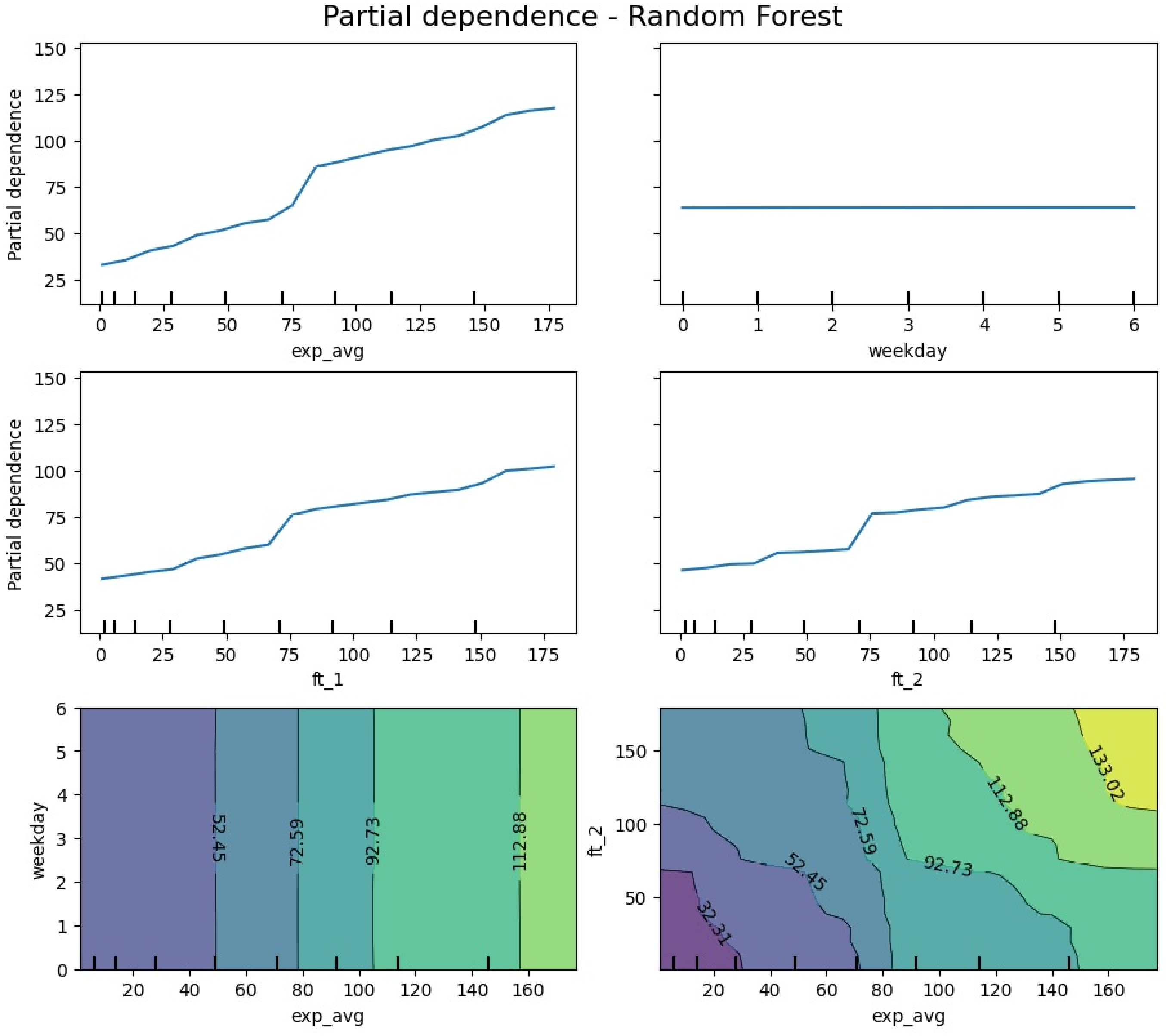
- Highly correlated features can confuse the interpretation of feature importance, dilute individual feature contributions, and lead to overfitting. This is also the case with models like Random Forest and XGBoost, addressing multicollinearity using dimensionality reduction, feature removal, or regularization can help improve model performance and interpretability.

## Random Forest Feature Analysis

In [18]:
```
from sklearn.inspection import PartialDependenceDisplay
```

In [19]:
```
common_params = {
    "subsample": 50,
    "n_jobs": 2,
    "grid_resolution": 20,
    "random_state": 0,
}

features_info = {
    # features of interest
    "features": ["exp_avg", 'weekday', 'ft_1', 'ft_2', ("exp_avg", 'weekday'), ("exp_avg", 'ft_2')],
    # type of partial dependence plot
    "kind": "average"
}

_, ax = plt.subplots(ncols=2, nrows=3, figsize=(9, 8), constrained_layout=True)
display = PartialDependenceDisplay.from_estimator(
    regr_rf,
    x_train,
    **features_info,
    ax=ax,
```

```
    **common_params,
)

_ = display.figure_.suptitle(("Partial dependence - Random Forest"),fontsize=16)
```




Q8 (5pts): Explain the partial dependence plot of 'exp_avg'.

- exp_avg has a upward trend, indicating that as 'exp_avg' increases, the predicted response (target variable) also increases. It has some steeper increases at certain intervals, which may suggest thresholds or points where changes in 'exp_avg' have a more significant impact on the predicted outcome. Clearly it is an important feature influencing the model's prediction. A rise in 'exp_avg' pushes the prediction higher.

Q9 (10pts): Is there any interaction between 'exp_avg' and 'weekday'? How about 'exp_avg' and 'ft_2'?

- According to the partial dependence plot, there are no interactions for exp_avg and weekday based on the contour plot. This is also supported by the flat line in weekday plot, where there is no significant influence. There is however interaction between exp_avg and ft_2 based on the contour plot on multiple levels.

In [20]:
```
!pip install shap
```

```
Requirement already satisfied: shap in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10/site-packages (0.46.0)
Requirement already satisfied: numpy in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10/site-packages (from shap) (1.23.5)
Requirement already satisfied: scipy in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10/site-packages (from shap) (1.10.1)
Requirement already satisfied: scikit-learn in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10/site-packages (from shap) (1.3.2)
Requirement already satisfied: pandas in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10/site-packages (from shap) (1.5.3)
Requirement already satisfied: tqdm>=4.27.0 in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10/site-packages (from shap) (4.66.5)
Requirement already satisfied: packaging>20.9 in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10/site-packages (from shap) (24.1)
Requirement already satisfied: slicer==0.0.8 in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10/site-packages (from shap) (0.0.8)
Requirement already satisfied: numba in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10/site-packages (from shap) (0.56.4)
Requirement already satisfied: cloudpickle in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10/site-packages (from shap) (3.0.0)
Requirement already satisfied: llvmlite<0.40,>=0.39.0dev0 in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10/site-packages (from numba->shap) (0.39.1)
Requirement already satisfied: setuptools in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10/site-packages (from numba->shap) (65.5.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10/site-packages (from pandas->shap) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10/site-packages (from pandas->shap) (2024.1)
Requirement already satisfied: joblib>=1.1.1 in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10/site-packages (from scikit-learn->shap) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10/site-packages (from scikit-learn->shap) (3.5.0)
Requirement already satisfied: six>=1.5 in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10/site-packages (from python-dateutil>=2.8.1->pandas->shap) (1.16.0)
```

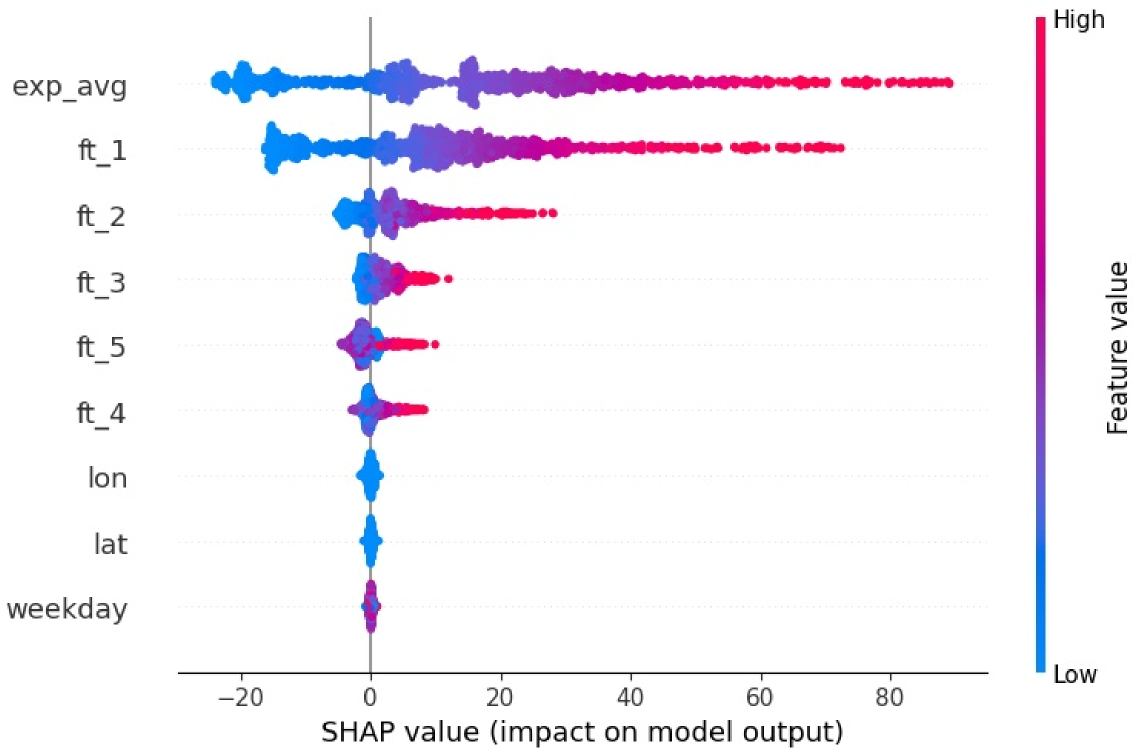
In [21]:
```python
import shap
```

In [22]:
```python
explainer_rf = shap.Explainer(regr_rf, x_train)
```

In [23]:
```python
shap_values_rf = explainer_rf(x_test[0:1000])
```

```
100%|===================| 995/1000 [00:37<00:00]
```

In [24]:
```python
shap.summary_plot(shap_values_rf, x_test[0:1000])
```

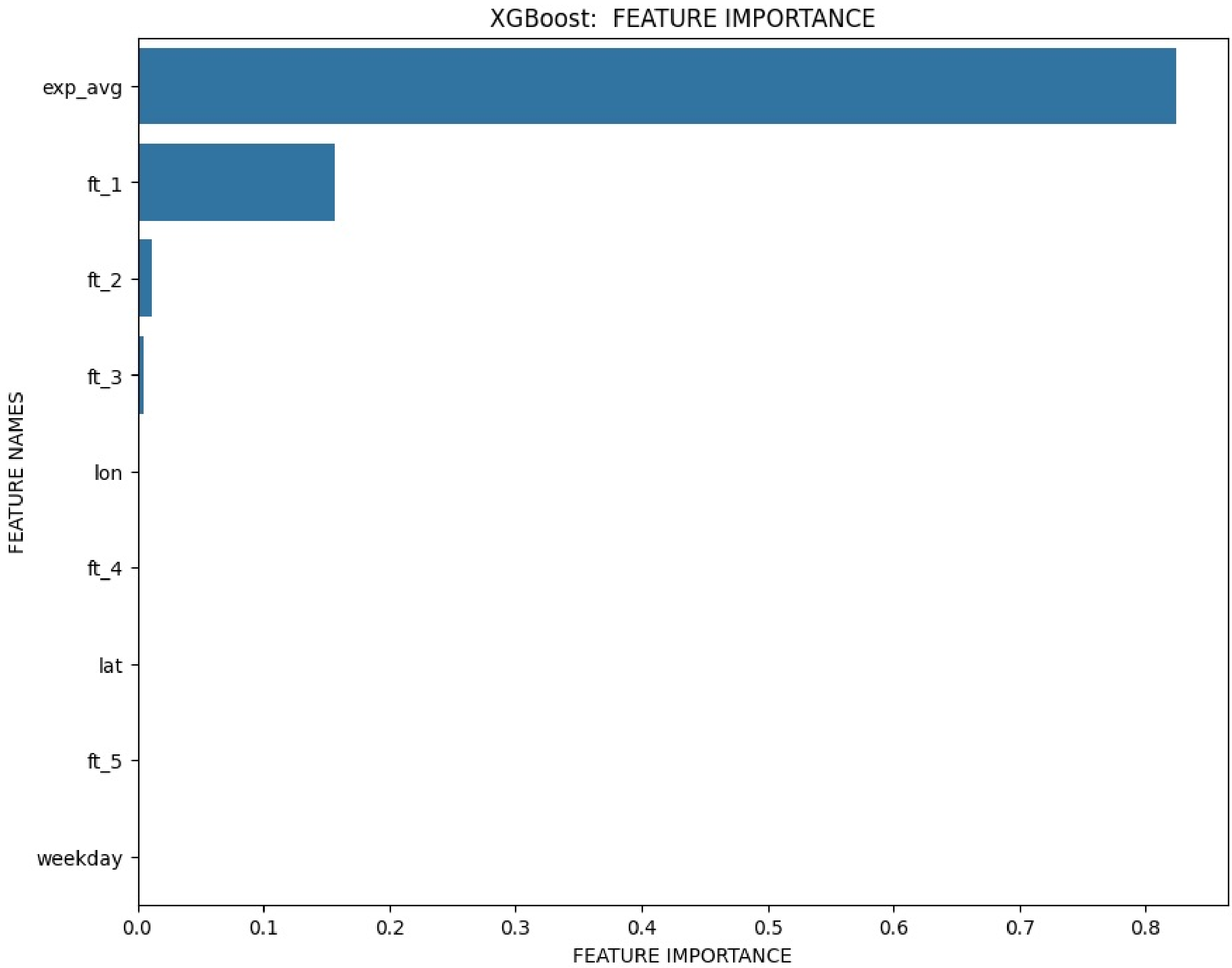



Q10 (5pts): Is `weekday` showing as an important factor on prediction explanations via SHAP?

- Weekday appears at the bottom of the plot with very small SHAP values that are clustered around zero. So SHAP is also showing that weekday is not an important factor on prediction.

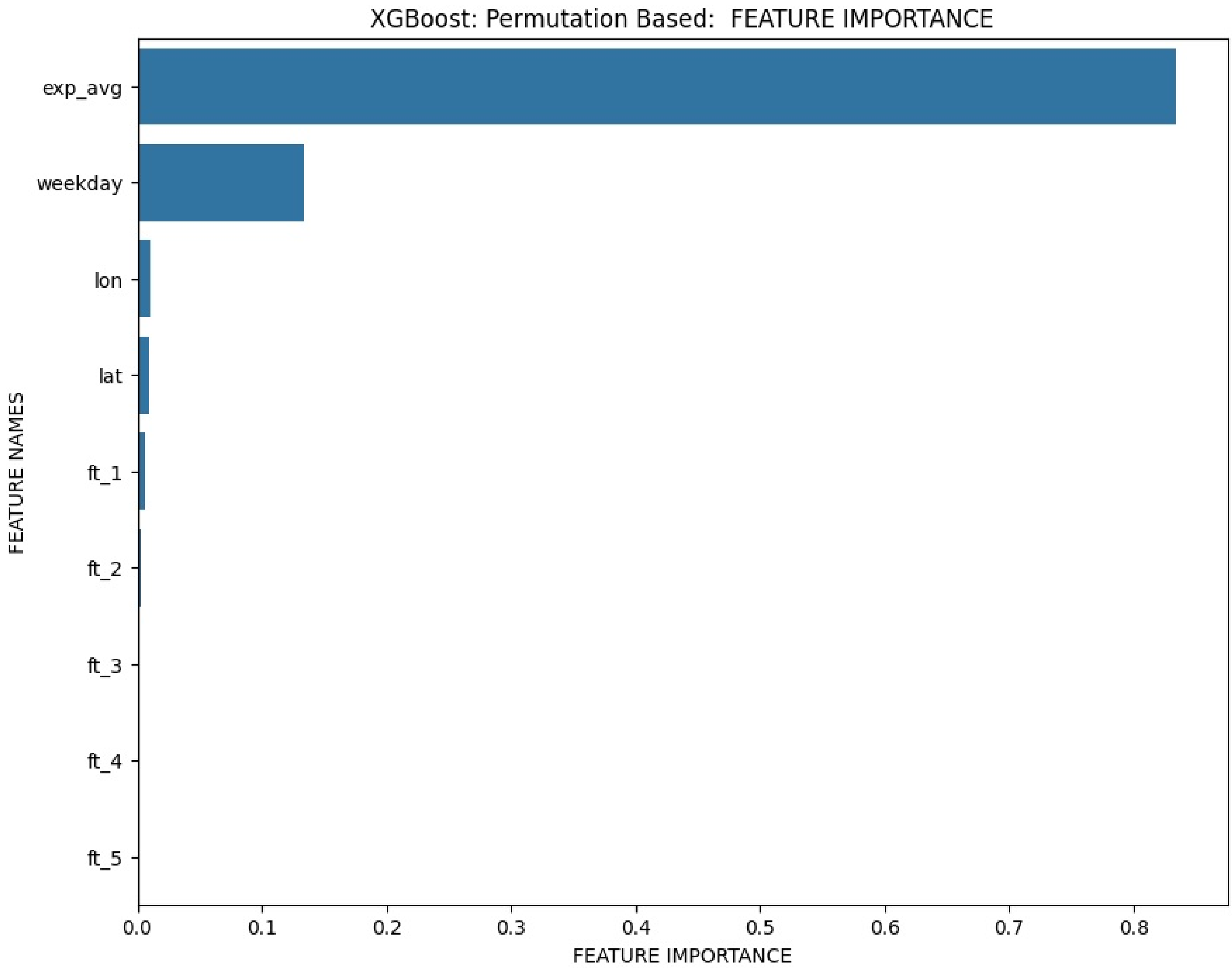
## XGBoost Feature Importance

In [25]:
```
plot_feature_importance(regr_xgb.feature_importances_,x_train.columns,'XGBoost')
```




In [26]:
```
permutation_based_feature_importance_xgb = permutation_based_feature_importance(x_train, y_train, mse_train_xgb

plot_feature_importance(permutation_based_feature_importance_xgb,x_train.columns,'XGBoost: Permutation Based')
```

```
Permuting feature  1
Permuting feature  2
Permuting feature  3
Permuting feature  4
Permuting feature  5
Permuting feature  6
Permuting feature  7
Permuting feature  8
Permuting feature  9
```



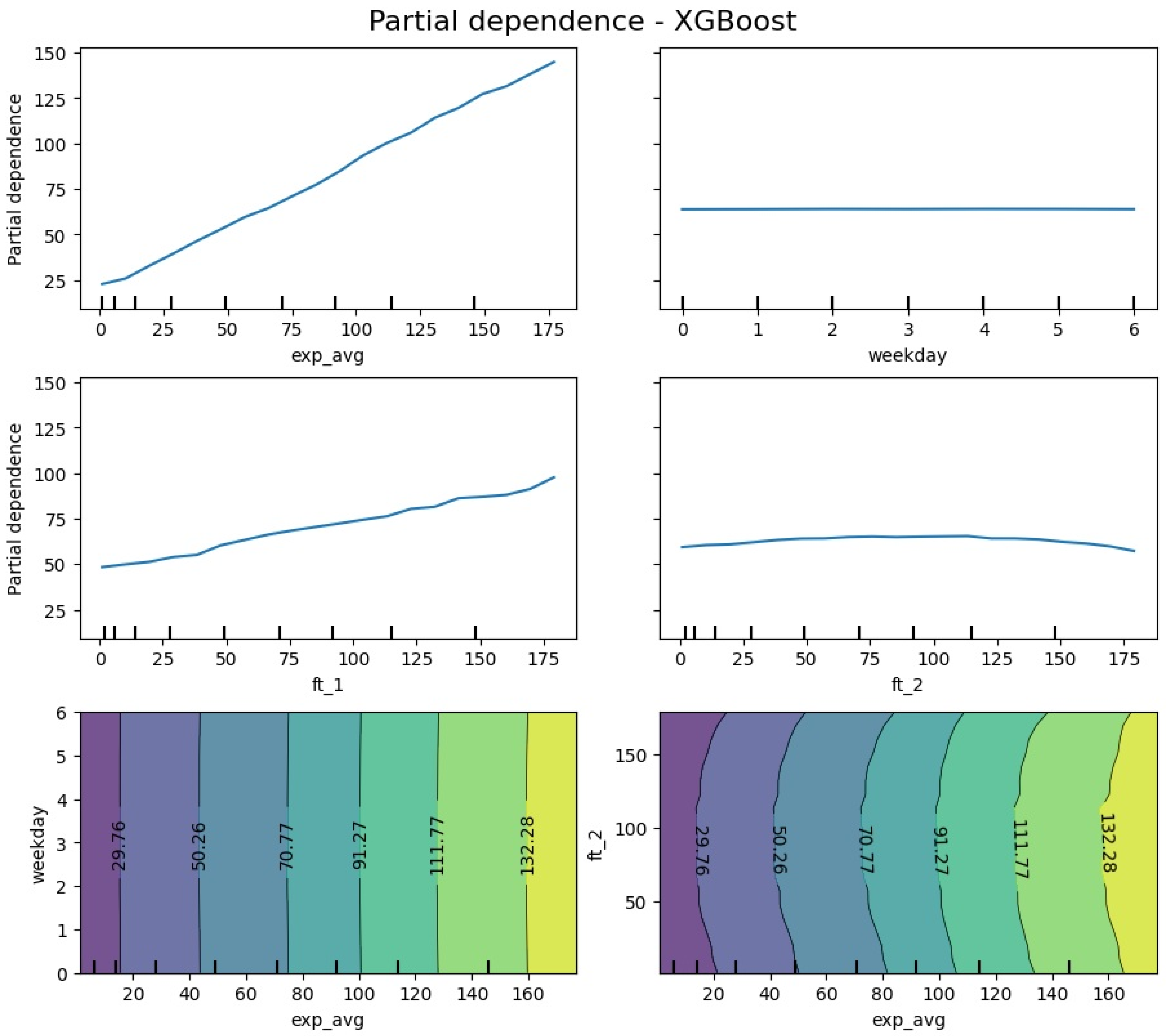

Q11 (10pts): Based on the permutation-method feature importance chart for the XGBoost model, would you recommend that the model take out the less influential variables ft_1, ft_2, ft_3, ft_4, and ft_5 ? Why or why not?

- Depends if you want to simplify the model without sacraficing the predicting power, overall the permutation plot shows that the ft variables have very little importance and contribute to nothing, leaving little to no change.

## XGBoost Feature Analysis

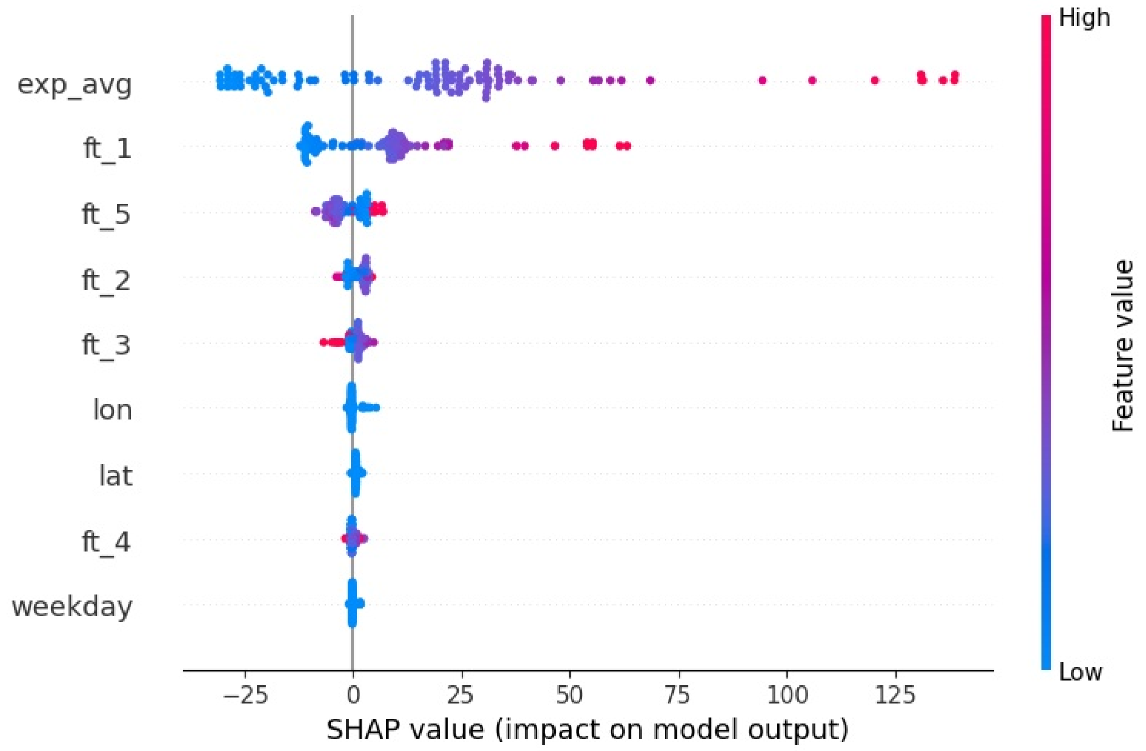
In [27]:
```
_, ax = plt.subplots(ncols=2, nrows=3, figsize=(9, 8), constrained_layout=True)
display = PartialDependenceDisplay.from_estimator(
    regr_xgb,
```

```
    x_train,
    **features_info,
    ax=ax,
    **common_params,
)

_ = display.figure_.suptitle(("Partial dependence - XGBoost"),fontsize=16)
```




In [28]:
```
explainer_xgb = shap.Explainer(regr_xgb, x_train)
shap_values_xgb = explainer_xgb(x_test[0:100])
```

In [29]:
```
shap.summary_plot(shap_values_xgb, x_test[0:100])
```

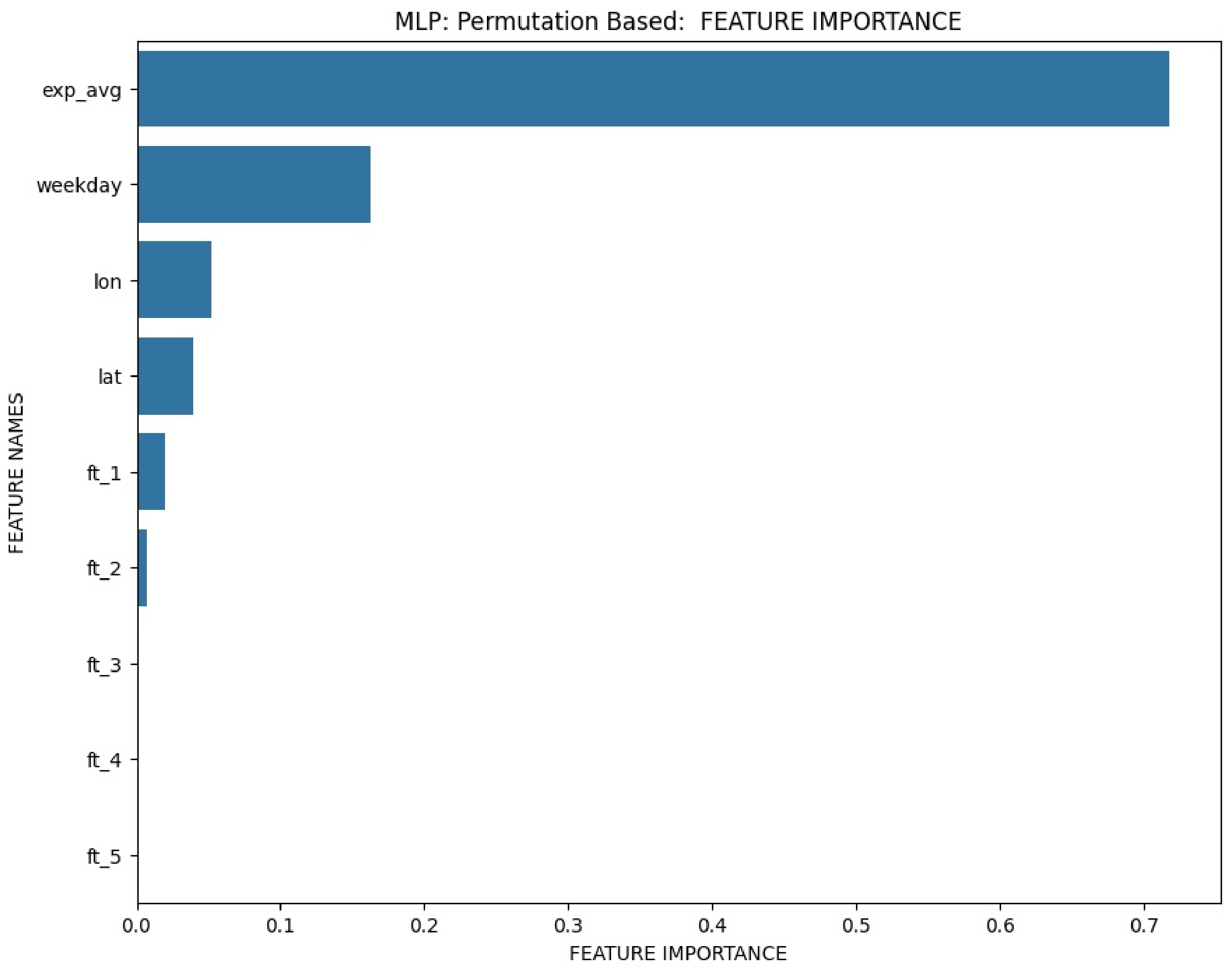



## MLP Feature Importance

In [30]:

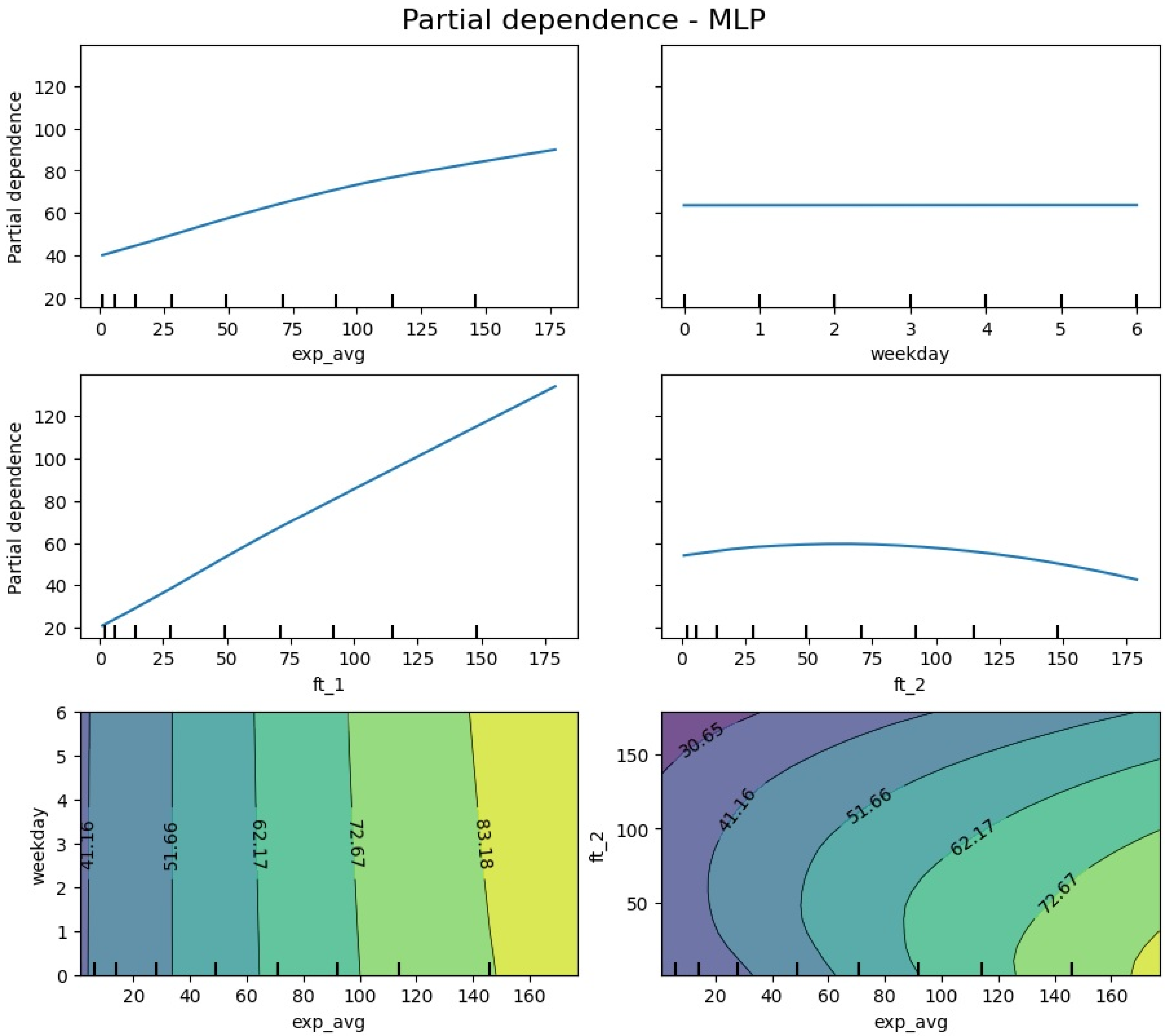
```
permutation_based_feature_importance_mlp = permutation_based_feature_importance(x_train, y_train, mse_train_mlp

plot_feature_importance(permutation_based_feature_importance_mlp,x_train.columns,'MLP: Permutation Based')
```

```
Permuting feature  1
Permuting feature  2
Permuting feature  3
Permuting feature  4
Permuting feature  5
Permuting feature  6
Permuting feature  7
Permuting feature  8
Permuting feature  9
```




## MLP Feature Analysis

In [31]:
```
_, ax = plt.subplots(ncols=2, nrows=3, figsize=(9, 8), constrained_layout=True)
display = PartialDependenceDisplay.from_estimator(
    regr_mlp,
    x_train,
    **features_info,
    ax=ax,
    **common_params,
)

_ = display.figure_.suptitle(("Partial dependence - MLP"),fontsize=16)
```

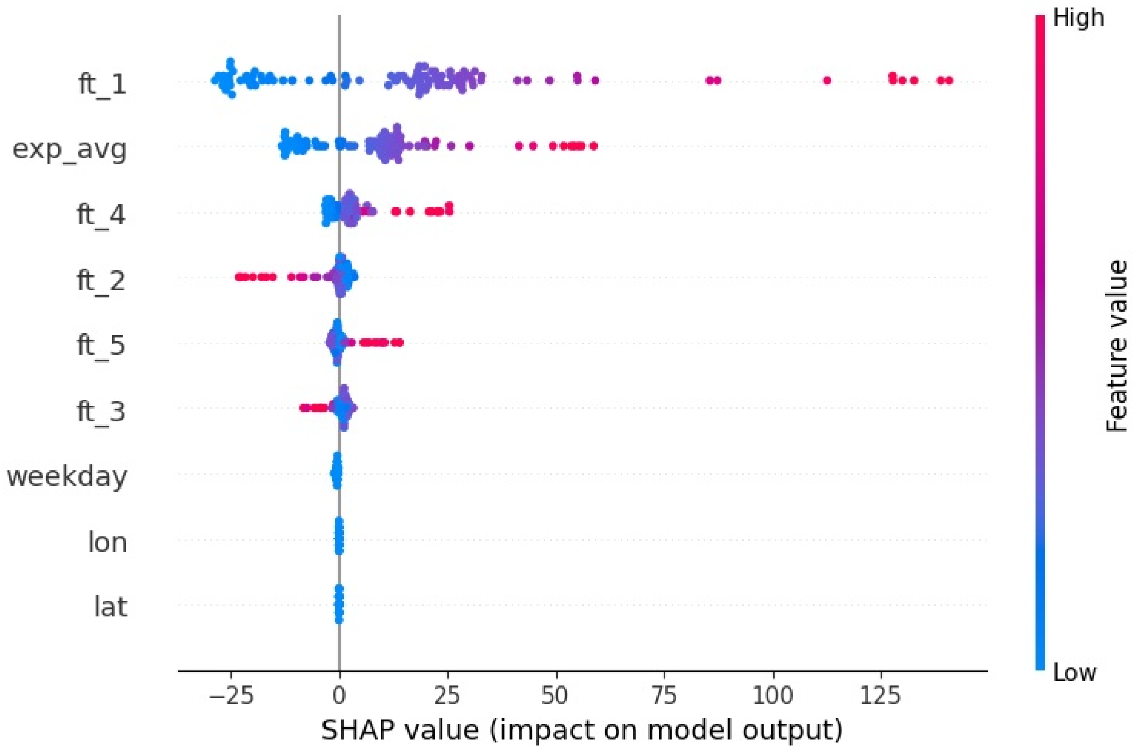



In [32]:
```
# for NN, we use 'KernelExplainer', but it's very very slow. So we use 'shap.sample' to sample a subset.
n_samples = 100
explainer_mlp = shap.KernelExplainer(regr_mlp.predict, shap.sample(x_train, n_samples))
```

In [33]:
```
shap_values_mlp = explainer_mlp(x_test[0:100])
```

```
 0%|          | 0/100 [00:00<?, ?it/s]
```

In [34]:
```
shap.summary_plot(shap_values_mlp, x_test[0:100])
```




## Local Explainability

### Random Forest SHAP

In [35]:
```
x_test[:1]
```

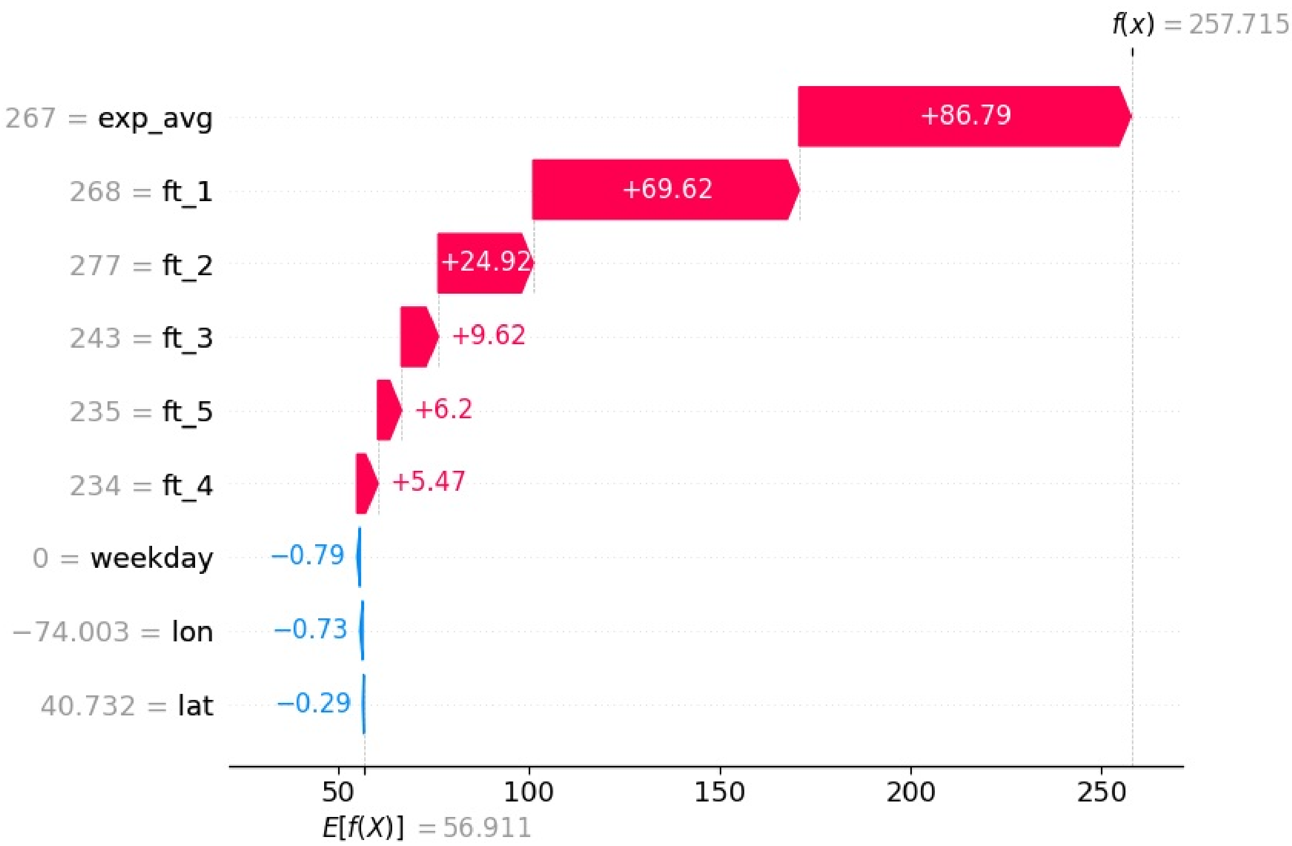
Out[35]:

| | ft_5 | ft_4 | ft_3 | ft_2 | ft_1 | lat | lon | weekday | exp_avg |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 235 | 234 | 243 | 277 | 268 | 40.732211 | -74.003238 | 0 | 267 |

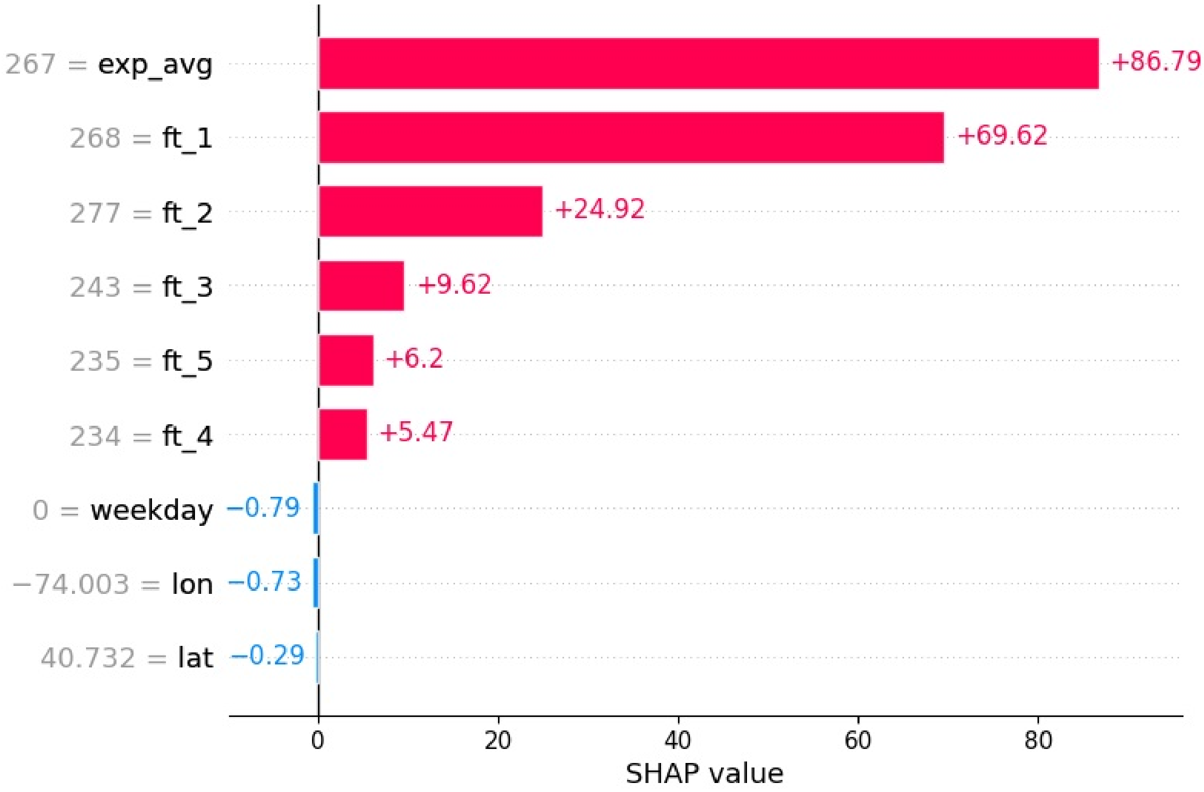
In [36]:
```
print('Actual:', y_test['target'].iloc[0], 'Random Forest:', y_pred_rf[:1], 'XGBoost:', y_pred_xgb[:1], 'MLP:',
```

```
Actual: 247 Random Forest: [257.71486111] XGBoost: [272.77777] MLP: [261.09911755]
```

In [37]:
```
shap.plots.waterfall(shap_values_rf[0])
```



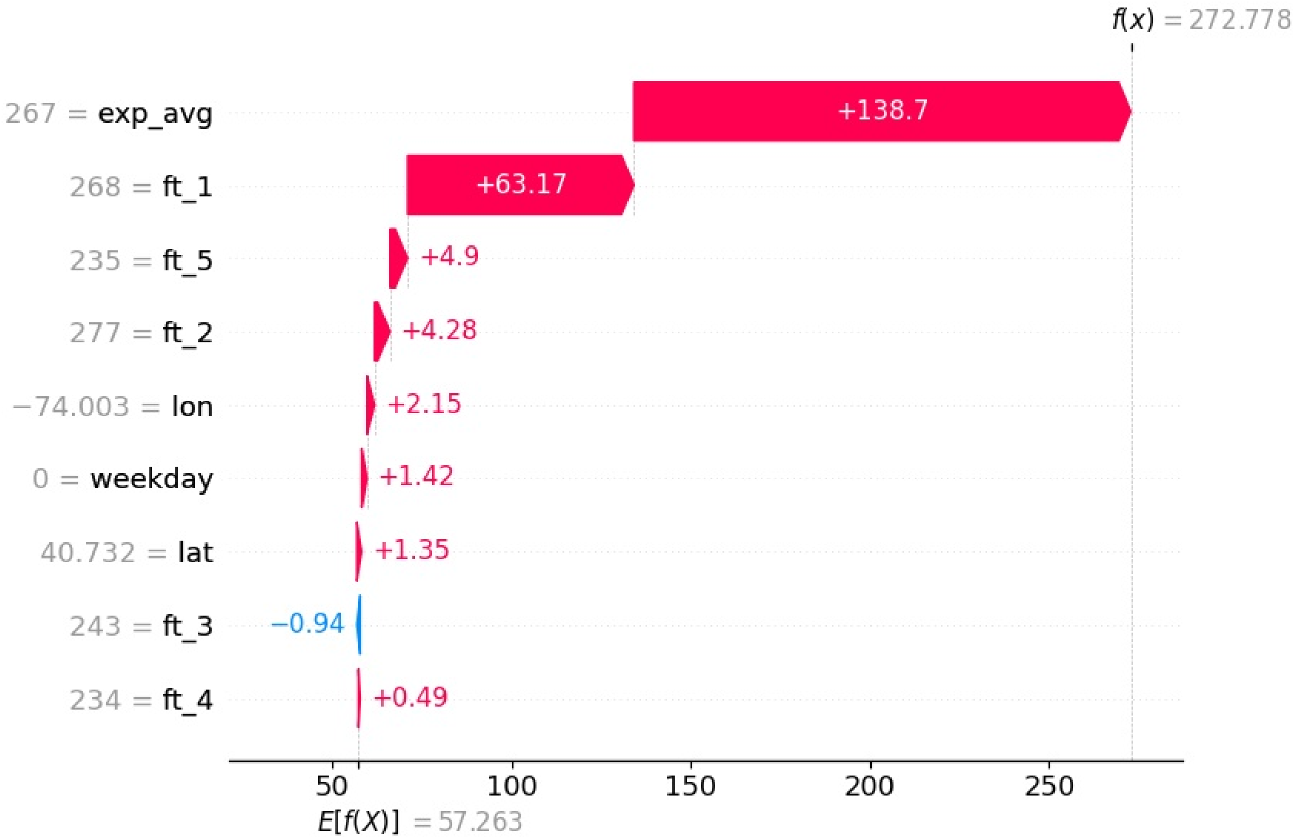
In [38]:
```
shap.plots.bar(shap_values_rf[0])
```



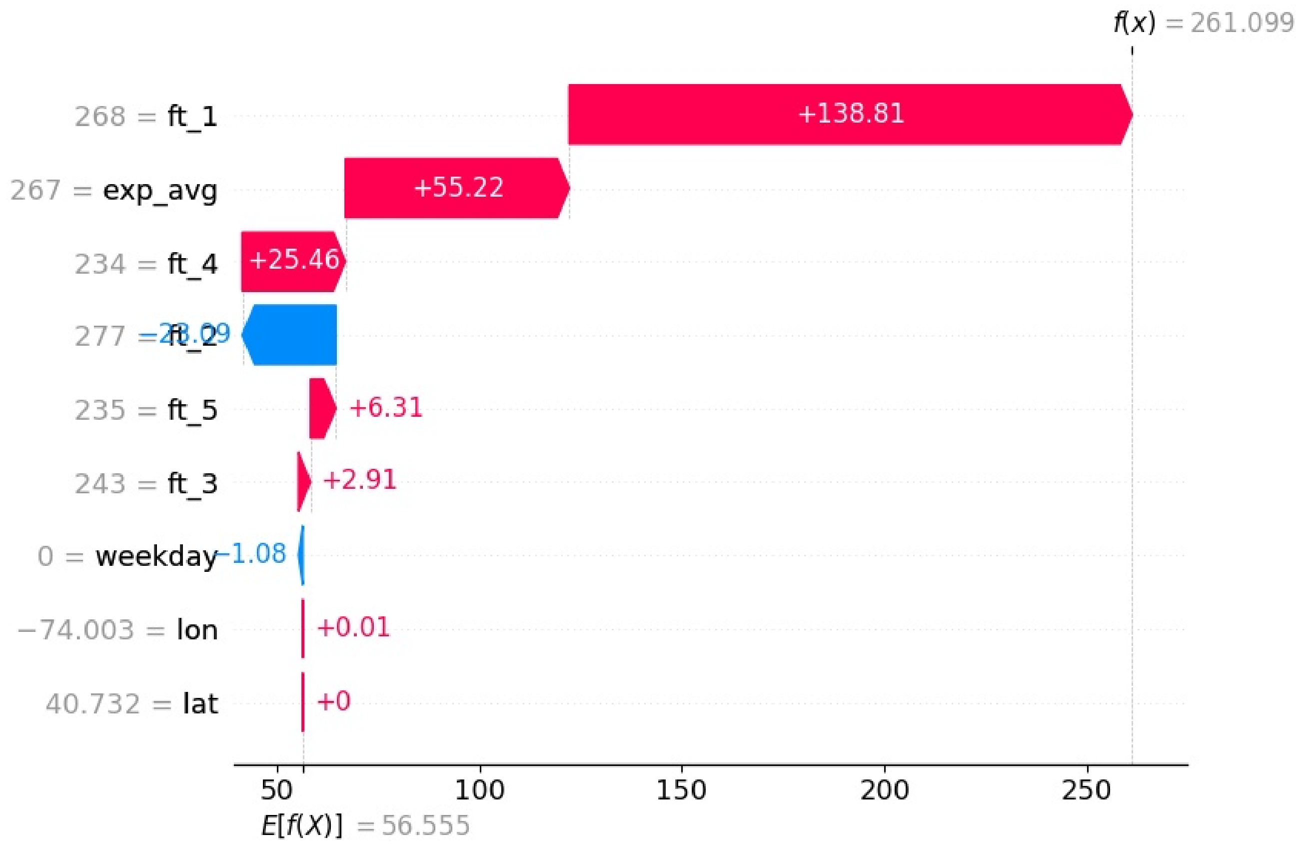

Q12 (5pts): What is the difference between the plots shown by `shap.plots.waterfall` and `shap.plots.bar`?

- Waterfall explains one single prediction point compared to the bar which shows the feature importance across the whole dataset. Waterfall is designed to make a specific prediction for one single instance. Bar plot shows the average SHAP value with most important features at the top.

In [39]:
```
shap.plots.waterfall(shap_values_xgb[0])
```




In [40]:
```
shap.plots.waterfall(shap_values_mlp[0])
```




Q13 (5pts): How is it possible that two models (like the XGBoost and MLP models above) can have very similar permutation feature importance but very different SHAP explanations for the same data point?

- Permutation feature is designed to focus on the global contribution of features across the data, while the SHAP explainations are for the local contribution. It reflects the differences in the XGBoost and MLP process feature interactions.

In [41]:

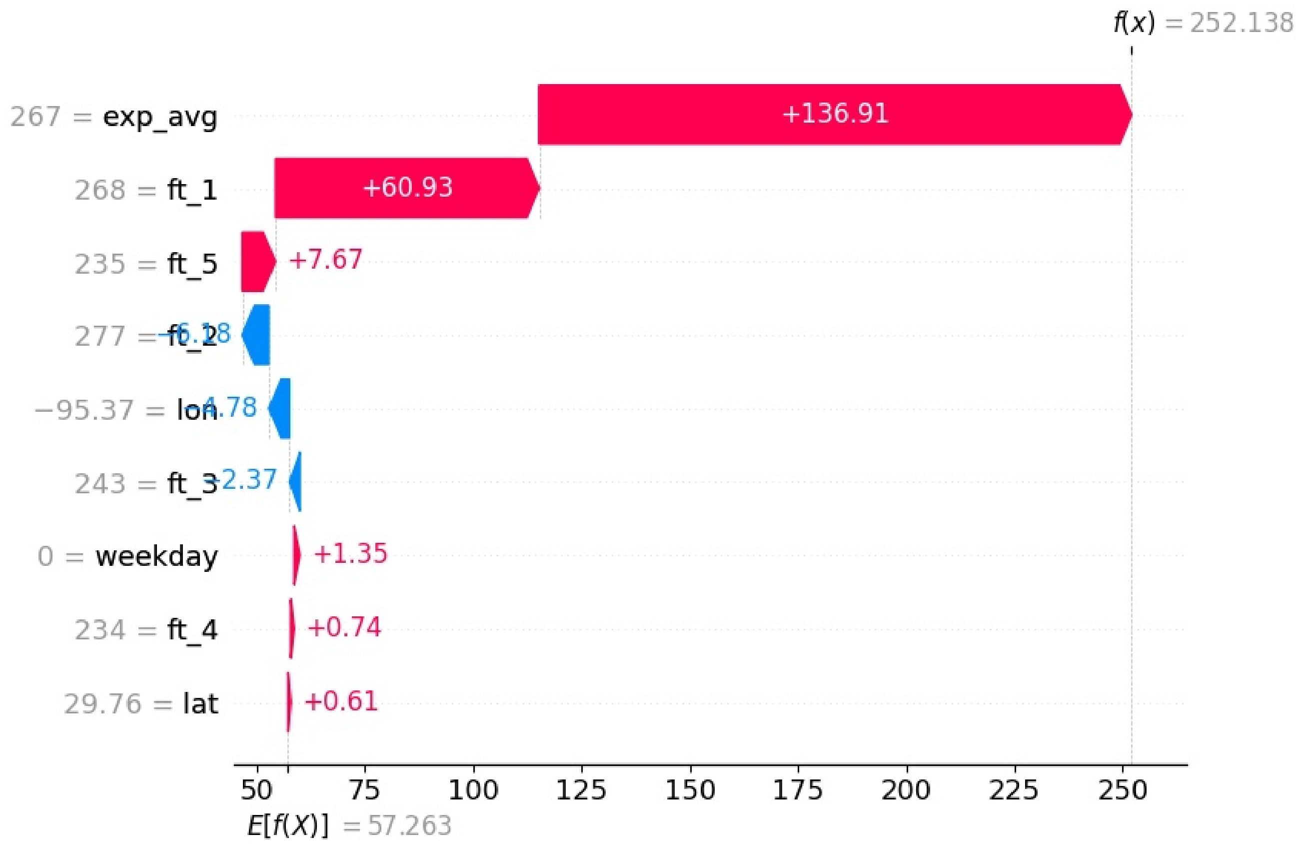
```
test_prediction = x_test[0:1].assign(lon=-95.369804,lat=29.760427)
test_prediction
```

Out[41]:

| | ft_5 | ft_4 | ft_3 | ft_2 | ft_1 | lat | lon | weekday | exp_avg |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 235 | 234 | 243 | 277 | 268 | 29.760427 | -95.369804 | 0 | 267 |

In [42]:

```
shap_values_test = explainer_xgb(test_prediction)
shap.plots.waterfall(shap_values_test[0])
```




Q14 (5pts): Are feature effects independent from each other in our SHAP XGBoost explainer?

- They aren't entirely independent from each other, SHAP inherently accounts for how features interact with one another and shows the marginal contribution of each feature. SHAP relects both the individual effect of each feature and in turn the influence they have on each other.

Q15 (5pts): `lon=-95.369804,lat=29.760427` is Houston, TX. What would a data scientist need to do to create good explanations for this region?

- Create partial dependence plots like earlier in the assignment, use domain knowledge of houston to help interpet the model better. Some preprocessing by reglarizing or feature selecting meaningful by looking at SHAP waterfall plots. For geospatial data of a specific area, knowing a general idea of you data and the conditions economically or housing wise goes far when interpreting models.